# Location Aware Code Offloading on Mobile Cloud with QoS Constraint

Kishwar Ahmed
School of Computing and Information Sciences
Florida International University
Miami, FL, USA
kahme006@fiu.edu

Md. Shakil Ahamed
School of Computing and Information Sciences
Florida International University
Miami, FL, USA
maham001@fiu.edu

Samia Tasnim
School of Computing and Information Sciences
Florida International University
Miami, FL, USA
stasn002@fiu.edu

***Abstract*— Mobile applications can be enhanced at a great extent by using the offloading mechanism in an energy efficient manner to the bounty resourceful clouds. Due to the great demand of smart phones, the issue of providing more processing capability to this resource constraint device is getting more concern now-a-days. In this paper, a method level off-loading mechanism has been described where no prior image of the mobile device is needed to be transferred to the cloud. The application is partitioned at different points where the migration of the execution thread is performed from mobile device to nearby resourceful cloud to get the best execution performance in optimal energy cost. The mobile can complete the execution after the partitioned thread returns back from the cloud to the device. The mobile cloud is able to provide service to the mobile devices independent of their personal configuration. This mechanism will be able to increase scalability as well as performance in the form of faster execution speed of the mobile devices. In addition to the above mentioned points, this paper also considers the mobility of the mobile device and proposes a solution on how to find the best cloud server on the move. To find out which cloud to off-load, the communication latency that can be inferred from the distance between the mobile and the cloud has not only been considered, the capacity as well as the current load at individual clouds are also considered to find out the best cloud to off-load to ensure the better service for the mobile device. The proposed solution has been simulated and compared against CloneCloud in two difference simulation scenerios.***

***Index Terms*— mobile cloud, mobility, location management**

## 1. Introduction

Today's mobile devices are less resourceful but resource hungry at the same time. With the increase of network resources namely high performance clouds; it is constantly investigated to find the optimal partitioning between the loads of mobile device and the cloud. Much of the recent research has been focused on different aspect of offloading. Along with offloading, the location of the servers used for computational benefits is also important. In many cases the latency to transfer adversely affects the performance of the offloading mechanism. We investigate that, the communication latency between the device and the cloud, and the current service load of the cloud is a factor in making a offloading decision. CloneCloud[1] provides a solution of code offloading by using per device clone in the cloud. MAUI[11] also provides a similar solution. CloneCloud divides the code into partitions and uses static analysis and dynamic profiling to build decision trees to make the offloading decision. We observed that, in this way, it could take a significant number of runs to gather sufficient information to make a optimal choice. It also incurs the cost of sending the codes to the cloud for profiling, which may be significant in low bandwidth scenarios.

In our work we focus on the network latency and relative distance between the device and the service provider. Limiting the latency of communication is of main importance. We also consider the mobility of the mobile device, so that the device can communicate with the nearest possible server in terms of cost. We propose a solution to use the existing network topology to keep track of the nearest cloud servers in a particular location and use the existing mechanisms like hand-off to trigger our proposed location based server selection algorithm. We take three step approach to reduce this cost and keep this minimized throughout the service period. First, we suggest multiple servers in strategically placed locations to effectively minimize the latency of device to cloud communication. In later chapter (3), we explain how we can use the existing network topology to select and maintain an active connection with the server. Second, we propose platform independent service at the cloud, which will increase the number of services, and provide privacy, as we don't keep any prior image of the mobile device. Third, we propose to build the cost tree for the different partitions of an application by comparing the current server capacity vector (SCV), with the mobile capacity vector(MCV), which will result in reduced cost in terms of data transfer as well as immediate decision making. SCV is a set of constraints like CPU speed, memory, energy consumption which determines the performance capability of the Cloud server. MCV is a similar set of constraint for the mobile device .

We also propose an algorithm to efficiently calculate the communication and execution latency of different partitions of a code to be executed on the service cloud and depending on the total cost decide whether to execute the service on the cloud or not. We consider here both the transmission cost of the service to the cloud and from the cloud to the mobile device to consider the overall communication cost. Moreover, we consider the QoS constraint and power consumption too, for educated guess on which platform would be suitable for service code execution. We simulated our proposed solution using ns-2 and compared it with the performance of CloneCloud. The comparison result shows that our Location Aware offloading method performs better than CloneCloud.

We organize our paper in the following manner: in chapter 2 we discuss our work with related contemporary work, in chapter 3 we discuss the different part of our system and define an algorithm for the offloading, in chapter 4 we discuss the simulation setup and in chapter 5, the performance evaluation has been discussed. In chapter 6 we provide possible future work. In the chapter 7 we discuss how our work contributes towards less cost in offloading in terms of time.

## 2. RELATED WORK

Mobile image offloading to the cloud server depends on some QoS (e.g., energy, delay etc) constraints along with some other persistent issues like network connectivity, location management etc. [5] considered intermittent connectivity of mobile devices where the mobile device uses the resource of other devices which are located near to it over a WIFI network. There has been some good amount of work on privacy related work on offloading to cloud server (e.g., [3], [6], [7]). However, there is another significant line of work where authors have discussed efficient way of detecting malware of the mobile devices on the cloud and thus decrease the computational overhead that may incur if done on the mobile device itself. In [6], record and replay method has been used. The full image of the device is sent to the cloud initially. All the system calls are performed in the cloud using an emulator in the same manner as it is performed in the mobile device. The problem here is that it takes huge time in the context switching between user and kernel region. This method is too conservative by allowing strict locking protocol to prevent concurrent access. By allowing concurrent read access, the performance can be improved. In [10], the installed app is uploaded from mobile device to the cloud server where the scanning is performed using third party tools like: Kaspersky, VirusChief etc. Uploading apps from mobile device and scanning them takes long time, which results in good amount of wait for the user to get the result. Thus the benefit of cloud cannot be achieved in the full phase. Moreover, it does not ensure security between the mobile and the server. The limitation on the maximum size of the application 20MB also a shortcoming of this method. Possible improvement can be achieved by building a secure channel between the device and the cloud, keeping a service running at the server which could continuously crawl the web for apps and make an index of malicious and safe Apps, without client request, automated cache update, provide the URL (not the whole app) to the server when installing new applications; this reduces the amount of time for the total process. Moreover, to gain faster access to the malware repository a distributed server can be used.

The decision making process of whether to offload the service to the cloud or not depends on some QoS constraints and also power usage according to [8], [9]. [4] discusses the decision issue of whether to offload or not depending on Bandwidth issue where they have considered two types of clone on the cloud: off-clone and back-clone. ThinkAir [2], device specific clone is not needed. They used a single server to accommodate all the requests.

There are different ways of offloading discussed in past literature. For example, in [6] the whole image of smart phone is transferred to the cloud, while according to [1] part of smart phone image is transferred keeping the rest of the part inside the mobile for further computation. In [11] the code is offloaded to the cloud which is another significant part of literature representing image offload to the cloud.

[12] is an extension of CloneCloud which has mainly improved on the intermittent connectivity issue of CloneCloud. Moreover this paper has proposed a solution on the decision making process of how much of a whole service to be transferred to the cloud depending on the tradeoff of completion time and service migration time between cloud to cloud. Another line of work related to taking decision on distribution of cloud geographically is [13], where the authors have proposed a solution on service distribution among different clouds, focusing mainly on time critical events.

However, to the best of our knowledge, no work has been done on offloading of service code considering location change of the mobile device and also depending on the partitioning of the services. Although, [1] discusses issue of code execution on the cloud where part of the code is executed on the cloud and depending on the performance there rest of the code either executes on the mobile device itself or back on the cloud.

## 3. ARCHITECTURE

In our system, we propose to use application independent clone servers which are located in geographically distributed manner. Clone server distribution could also be designed according to the client request load. By application independent, we mean that no prior per device image is needed to be stored in the server. Request from the mobile device will be served independent of the configuration of the mobile device.

To incorporate location change of the mobile device, before offloading, the cloud server which demands less cost in term of communication latency and possible execution time will be chosen. Depending on the current location of the user the service execution platform may need to change. For example, a user may be currently in Miami and he/she may change current location to New York. Offloading a service to new cloud platform near New York would incur less cost than executing on the fixed cloud server for example at Miami which is the common case where the image of the mobile device is kept in a fixed server like in CloneCloud, MAUI etc.

*A. Overall Design:*

*Location Change Decision Module (LCDM):*
Whenever there is change of location of a MD in terms of location area i.e. it goes from one location area to another, this module helps to take decision on whether to continue using the previous cloud for service execution or to assign a new cloud server for the new location. For example, whenever a MD goes away from a current location area to a new location area, it would be obvious that executing the service at a server located closest to that location would be good choice. However, the mobile device should have the knowledge of the nearest cloud servers to make the decision of changing the active server. Here, we propose to use the existing network topology to provide the MD with the list of nearest servers. We suggest that, VLR of the location area will keep a list of nearest cloud servers. We suggest an algorithm to keep a list of possible servers in the MD. The hand-off mechanism of the current 3G networks will work as a trigger to our algorithm. When the MD changes location area, and hand-off is initiated, network will send the list of nearest servers to the MD.

*Profile Tree Building Module(PTBM):*
This module takes as input the SCV's of the cloud servers from the native list. It calculates the the new profile tree depending on the current status of the application , i.e. which nodes are still in the server for computation, which nodes are already done, etc. This module is a part of the LCDM.

*Service Migration Decision Module (SMDM):*
This module is responsible for taking decision on whether to transfer the service to the cloud based on service profile tree. We also take decision based on the QoS (e.g., latency, bandwidth etc) along with power constraint. While considering latency as decision criteria we take into consideration communication latency along with service latency on the cloud.

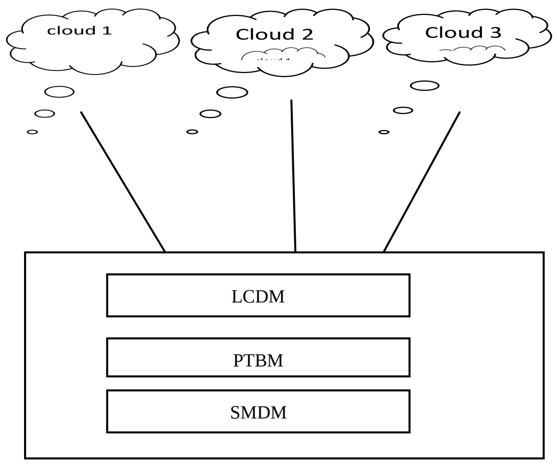

Fig 1: Overall Architecture Design

We briefly describe the responsibility of each modules here.

*B. Updating the neighboring cloud server list(LCDM)*

We define the following key terms for our algorithm:
Active cloud server Acl: the cloud server currently serving the MD's requests.
Native list of neighboring cloud NatNcl: current list maintained in the MD for nearest cloud servers
Current list of neighboring cloud CurNcl, current list provided by the VLR when a hand-off is initiated.

*Algorithm 1: Update Native CloudList*

```
begin
        If($NatN_{cl} == CurN_{cl}$)
                No calculation required
        Else if ($NatN_{cl} \subseteq CurN_{cl}$)
        begin
                Calculate $S = CurN_{cl} - NatN_{cl}$
                for each $S_i \varepsilon S$
                        Calculate gain factor for $S_i$
                CreateProfileTree();
        end
        Else if ($CurN_{cl} \subseteq NatN_{cl}$)
                set $NatN_{cl} = CurN_{cl}$
end
else
begin
        for each $S_i \varepsilon CurN_{cl}$
                Calculate gain factor for $S_i$
        set $NatN_{cl} = CurN_{cl;}$
        CreateProfileTree();
end
```

To briefly explain the algorithm, when the $CurN_{cl}$ is sent from the network, it is checked if $A_{cl}$ is present in the list.If it is present then, $CurN_{cl}$ is compared with the $NatN_{cl}$. if both are same then no new calculation is needed. If $CurN_{cl}$ is a subset of $NatN_{cl}$, the absent cloulds in $CurN_{cl}$ is removed from $NatN_{cl.}$ If NatNcl is a subset of CurNcl, calculate gain factor for all the new clouds in the $CurN_{cl}$. If none of the servers in $NatN_{cl}$ is present in the $CurN_{cl}$ then $NatN_{cl}$ is replaced by $CurN_{cl}$.

*C. Building the profile tree(PTBM)*

Instead of executing all the methods in the clone to build the execution trace for clone, we initially

- Collect the current SCV from the nearby clone servers. This is a single step process which only requires the server to send its SCV to the client until the next notification time. This notification time depends on the server load change. If there is a drastic change in the server load, immediate notification about the current SCV is sent to the client. Else, the SCV update is provided to the client in periodic manner. Server sends the SCV to the mobile device, only if the device has some outstanding task in the server at that time, e.g. the device is still connected for service. Sending the SCV when server load changes drastically will help the mobile device to build a new profile tree to make optimal decision regarding choice of the server.

- Use the MCV to compute the gain factor by which the offloading will improve the performance.
- The profile tree for the clouds will be estimated simply by multiplying the gain factors to the native profile tree.

This profile tree is used in the next algorithm along with QoS constraints to find optimal offloading decision.

## *D. Identifying the nodes and cloud server for migration:*

In our model, we denote the service code to execute on the cloud by S. We partition the service code into N parts, denoting each part of the service code by $S_i$, where $i\varepsilon\{1,2,\ldots,N\}$. These partitions are actually methods of the executable similar to [1]. Now to dynamically transfer a part of a service , we take into consideration the cost of migration ($m_i$) along with the service execution cost ($E_i$). Ei can be found from the profile tree built for each server. Moreover we also take decision based on QoS constraint (energy, latency etc). We denote the mobile device by MD and Cloud Server where the service is to be executed by CL and take decision of service execution on either of these places.

In algorithm 2, we also take into account the communication latency and service latency of service at the cloud server. Moreover, while taking input of parameters for each service we also take input the server parameters. For example, we devide the cloud servers into M parts denoting them by C1, C2, ..., Cm and characteristics of each server part are taken as input while taking the migration decision. Based on these notations we present the following algorithm to take decision on service migration to the cloud:

*Algorithm 2 : Cloud Server Selection*

```
1.Begin
2. Partition S into N parts: $S_1,S_2,...S_N$
3. If input parameter given for QoS
        Calculate QoS.S |MD and QoS.S |CL based on given input
4. Else Calculate QoS.S |MD and QoS.S |CL based on past history
5. Calculate $m_i$ and $E_i$ both at MD and CL
6. Find Cost|MD and Cost|CL where Cost|MD = $m_i$|MD + $E_i$ |CL
7. If QoS.S |MD < QoS.S |CL and Cost|MD < Cost|CL
Execute the code partitions in the Mobile
8. Else execute the code in the Cloud Server
9.End
```

As input parameter this algorithm takes the following:

- Configuration Parameter: These parameters represent those classes of inputs which can be varied and configured according to the choice of algorithm used for the service. For example, for a cryptography algorithm, the size of the key can be a good example of this kind of parameter.
- Input Size: Example of these input values include size of the input image (for example, this value is large for high resolution image and so on).
- Platform Environment: This input value represent the characteristic of the environment (either mobile device or the service cloud) for the decision maker to take decision which platform would be ideal choice.

The output of the proposed algorithm includes what platform to choose for service execution and that too at optimal cost value and location variability. However there are some tasks which can not be migrated to the cloud server and for those types of services we do not consider the action needed to be taken for task migration. These types of services are present for both Cloud and mobile devices [15], [16].

## *E. Connecting to the new cloud:*

After the server selection is done, connecting to the new cloud and disconnecting the previous one is not a straight forward choice. Some of the nodes in the profile tree may still be executing in the old cloud, when the decision to connect to new cloud has been made. We consider the following options for smooth transition between the servers:

- Connection to the old cloud will be uncoupled. And the nodes which were being serviced at this time will be sent to the new server for computational benefit. The disadvantage of this scheme is that a lot of have to be repeated if multiple nodes were being serviced at the cloud at that moment.
- Connection to the existing server will be kept until the current serving nodes are returned to the MD. This way, the overhead of executing the same node multiple time is removed. But, the overhead of increased transmission delay remains in effect.
- Connection to the old server will be closed immediately, but address of the new server will be sent to the currently executing one before the connection is closed. The execution will go on on the disconnected server and will be sent to the new server after completion. In this way, the MD does not have to keep two active connections at the same time.

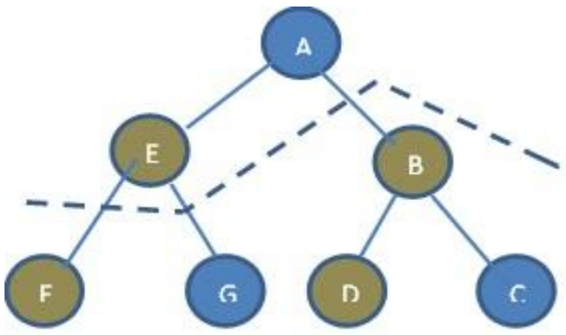

Fig 2: Profile Tree

As an example, we consider the above profile tree where blue nodes are executed on the MD and others are executed on the cloud. At this instance of the tree, node A and E are complete. But at the time of connecting to new server, node B is executing in the old server. So, we can disconnect the connection with old server immediately and send node B to the new server (option A), or keep the existing connection active until node B‘s execution is complete(option 2) or disconnect the old server and not send node B to new cloud, and wait for the new cloud to receive the result of node B’s execution from the old server and send it back to MD. Choice of options among the above three depends on the size and nature of the nodes. As we have not done evaluation on these models we cannot identify the best solution. But we prefer to keep two connections live at the same time (option b) to reduce overhead of multiple execution (option a) and overhead of cloud-to-cloud connection establishment (option b).

## 4. Simulation Setup

In this section, we describe the environment we created and different parameters we used for comparison with the algorithm presented in CloneCloud. Our simulation setup was built and tested using simulation software ns2.

We considered two scenarios for simulation where each of them had number of cloud servers, base station for each cell area and some mobile hosts moving from one cell area to another causing handoff to take place. Each of the parts of simulation setup along with the configuration parameter is described below:

- Cloud Server: Our algorithm uses cloud server positioned at different places for running of application depending on the workload requirement of that application. Whenever handoff takes place for a mobile host, we recalculate the distance between the mobile hosts and the cloud server and based on the distance find the nearest cloud with required capacity to execute the application. Instead of sending the whole code to the cloud server we however send a partial code to the server and thus optimize the execution of code at cloud or mobile hosts.

- Base station: In our simulation setup, we have base station at each cell area which controls all mobile devices under that cell area and takes decision of handoff whenever it takes place.

- Mobile hosts: We had some mobile hosts positioned in different cell area which moved from one cell area to another. These mobile hosts are responsible for execution of different applications on the service platform and offloading the tasks to the server.

The following parameters were measured for our comparison with the algorithm CloneCloud:

Table 1: Simulation parameters

| | |
|---|---|
| Channel | Channel/Wireless |
| Network Interface | Wireless |
| NS Version | Ns-2 |
| Interface Queue | Drop Tail |
| Queue Length | 50 |
| Simulation Area Size | 670*670 |
| Simulation Duration | 250 sec |
| Number of Mobile Node | 1 |
| Number of Base Station | 2 |
| Coverage of Base Station | 75 meter |

- Packet Size: We measured the packet size of each packet being transferred from a mobile host to a cloud server. Without loss of generality, we assume that these packet sizes represent the code size of target applications which need to be executed in the mobile hosts.

- Round Trip Time (RTT): Each packet is sent to the cloud server and executed while returning the output to the mobile hosts. We measure the RTT for each packet used in our simulation and used that information to measure the delay, which is another criterion for our comparison with other algorithm.

- Bandwidth: The channel bandwidth which signifies the present channel condition is considered and calculated in each of the simulation setup. We varied the bandwidth within a certain range to find out the effect of handoff and corresponding result on migration of code to the nearest cloud.

- Delay Time: For each packet to be transferred between a cloud sever and mobile hosts delay takes place which is a significant criterion to compare our algorithm with CloneCloud. Whenever there is handoff for a mobile host the delay varies depending on the position of the cloud server where the code is to be executed.

In the first topology (fig. 3), a mobile host(MH) moves from one location towards another cell area for the first 100 second. The movement of the MH begins in 100sec. After 200 sec, it returns towards the starting position and the simulation stops at 250 seconds. The simulation has been performed for both our proposed solution (Location Aware) and for Clone Cloud.

## 5. Performance Evaluation

This section outlines the improvement of our algorithm over other algorithm (CloneCloud) based on the simulation setup we have used. We compare on the basis of the following comparison criteria:

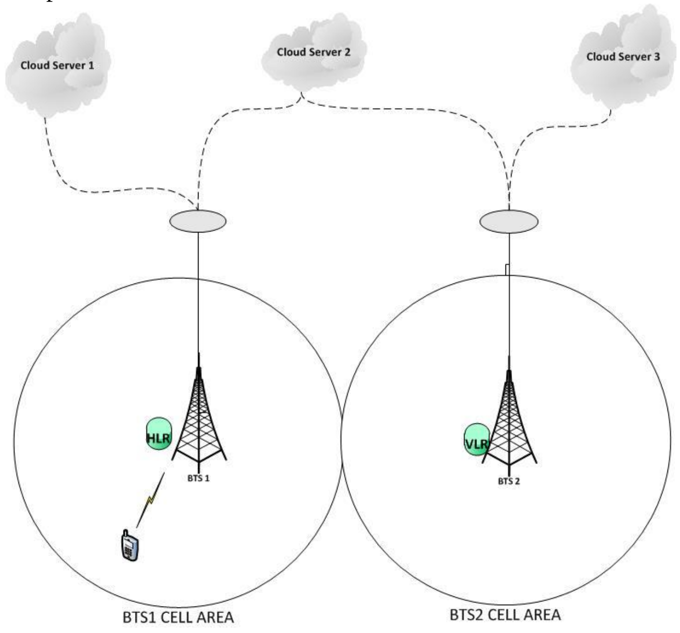

Fig 3: Simulation Scenerio for set up 1

- Throughput: This is a measurement of successful message transfer rate between a mobile host and cloud server. We calculate the byte transferred per second for both of the scenario we created and for both our algorithm and the algorithm used in CloneCloud.
- Delay: We measure the delay by calculating the average time it takes for a packet to reach the destination and return to the source. We thus compare both our algorithm and CloneCloud based on this parameter.
- Power: Another QoS constraint that we take as a base of comparison is power consumption by each mobile device when they move around different cell area. Different mobile device contains varying power consumption capability and whenever it transfers from a cell area to another, power consumption by that mobile device changes. Based on this observation, we compare algorithms to show how energy impacts them.

Now we compare the existing solution of CloneCloud for code offloading to the cloud server and our location aware code offloading with QoS constraint.

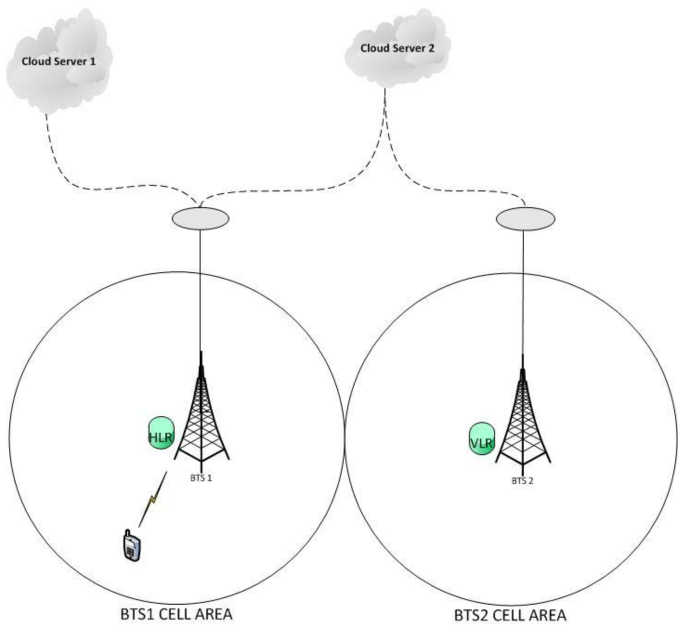

Fig 4: Simulation Scenerio for set up 2

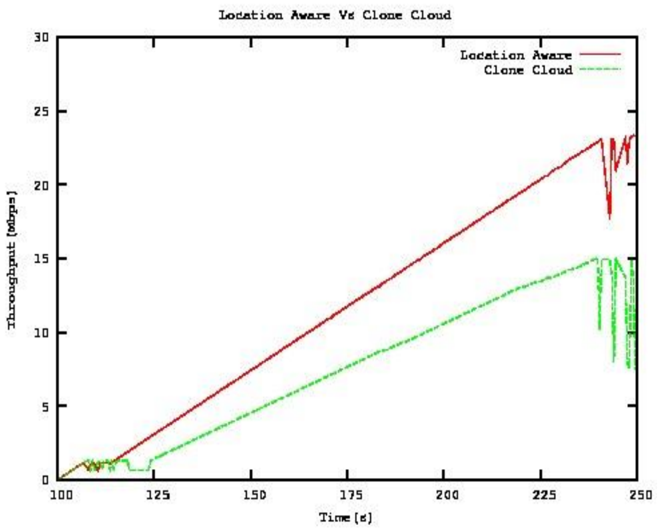

Fig 5: Throughput Comparison between CloneCloud and Location Aware Code offloading for topology 1

In the simulation scenario of fig 3, in the beginning, code offloading was performed to cloudserver1. Later when the MH moved to cell area 2, the current neighboring list was updated by cloud server 2 and cloud server 3. The load in server 3 was low as well as the channel bandwidth towards cloud server 3 was three times better than the link towards cloud server 1. Cloud 3 was nearer to MH in comparison to cloud 1 too. During the transition from cell area 1 to cell area 2, the selected cloud server for offloading was changed from cloud 1 to cloud 3 to ensure better throughput. On the other hand, in the CloneCloud, offloading is performed to the only server containing the image of the mobile device. Throughout the whole execution period, the code offloading was performed between MH and cloud server1.

Moreover, in figure 6, it can be seen that the overall delay in the communication between the MH and offloaded cloud is less in comparison to the CloneCloud. At time 125second, it can be seen that the difference is about 3 seconds.

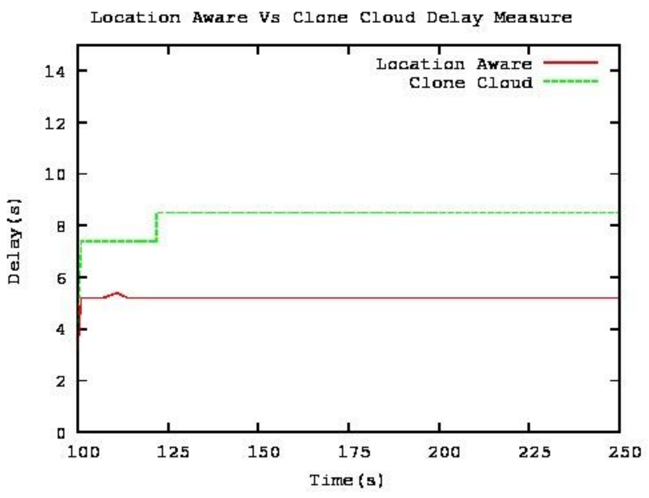

Fig 6: Delay Comparison between CloneCloud and Location Aware Code offloading for topology 1

In the simulation set up of fig 4, the set of neighboring cloud server is same for both the cell areas. The capacity and load of the cloud servers are identical. There is change in the channel bandwidth and transmission delay which influenced the Location Aware offloading method to choose cloud server 2 as the server to offload code. CloneCloud used cloud server1 which contain the image of the mobile host for performing code offload. Fig 7 and fig 8 depict the better performance of Location Aware method over CloneCloud.

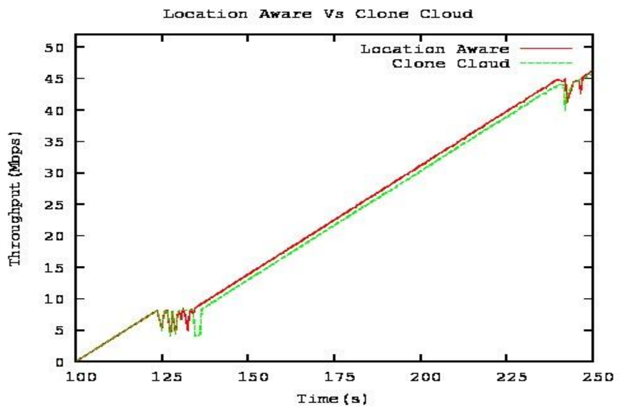

Fig 7: Throughput Comparison between CloneCloud and Location Aware Code offloading for topology 2

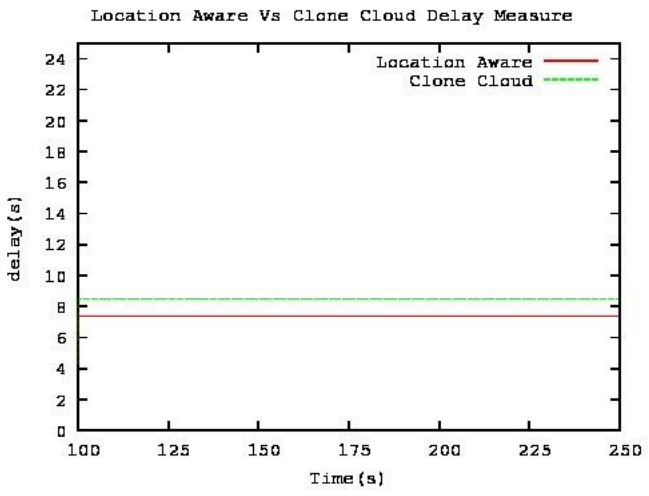

Fig 8: Delay Comparison between CloneCloud and Location Aware Code offloading for topology 2

## 6. FUTURE WORK

In this paper, we proposed solutions for dynamically provisioning services of mobile device to the mobile cloud based on service profile tree construction and location management of the cloud server. We simulated our proposed solution in ns-2 and showed that it performs better than the CloneCloud in perspective of throughput and delay. Although our proposed model provides a complete solution to the online service allocation to different servers along with QoS optimization, we could not consider some other constraints of mobile devices (for example, heterogeneity of network architecture etc). Moreover, there can be some privacy issues being introduced due to the implementation of our system (e.g., eavesdropping etc). We also did not consider non-migratable tasks (for example, tasks that must be executed at the mobile devices) and their impact on service platform selection algorithm which can be another important extension of our current work.

## 7. CONCLUSION

Our proposed system incorporates the location change of the mobile device before code offloading to a cloud server. Moreover, the extra cost of dynamic profiling by executing the process in the cloud has been avoided by using the concept of gain factor to estimate the execution time in the cloud. We simulated two experimental scenerios using Network Simulator 2 to compare the performance of our algorithm and CloneCloud. Instead of a fixed cloud, depending on the geo location of the mobile device, as well as the existing load and capacity of the nearby cloud servers, the choice of the cloud is made for code offload. This ensures better QoS which reflects in the simulation results.

## REFERENCES

.

[1] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, Ashwin Patti. CloneCloud: Elastic Execution between Mobile Device and Cloud. EuroSys Proceedings of the sixth conference on Computer systems 2011.

[2] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, Xinwen Zhang. ThinkAir: Dynamic resource allocation and parallel execution in cloud for mobile code offloading. INFOCOM, Proceedings IEEE 2012.

[3] Zhibin Zhou and Dijiang Huang. Efficient and Secure Data Storage Operations for Mobile Cloud Computing. Network and service management (cnsm), 8th international conference and 2012 workshop on systems virtualiztion management (svm). 2012.

[4] Marco V. Barbera, Sokol Kosta, Alessandro Mei, and Julinda Stefa. To Offload or Not to Offload? The Bandwidth and Energy Costs of Mobile Cloud Computing. In Proc. of IEEE INFOCOM, 2013.

[5] Cong Shi, Vasileios Lakafosis, Mostafa H. Ammar, Ellen W. Zegura. Serendipity: Enabling Remote Computing among Intermittently Connected Mobile Devices. MobiHoc'12, June 11–14, 2012.

[6] Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis, Herbert Bos. Paranoid Android: Versatile Protection For Smartphones. ACSAC Proceedings of the 26th Annual Computer Security Applications Conference 2010.

[7] Saman Zonouz, Amir Houmansadr, Robin Berthier. Secloud: A Cloud-based Comprehensive and Lightweight Security Solution for Smartphones. Computers & Security, 2013.

[8] Yunqi Ye, Nisha Jain, Longsheng Xia, Suhas Joshi, I-Ling Yen, Farokh Bastani, Kenneth L Cureton and Mark K Bowler. A Framework for QoS and Power Management in a Service Cloud Environment with Mobile Devices. Fifth IEEE International Symposium on Service Oriented System Engineering, 2010.

[9] Y Ye, L Xiao, IL Yen, F Bastani. Leveraging Service Clouds for Power and QoS Management for Mobile Devices. Cloud Computing (CLOUD), 2011 IEEE International Conference, 2011.

[10] Chris Jarabek, David Barrera, John Aycock. ThinAV: Truly Lightweight Mobile Cloud-based Anti-malware. ACSAC'12 Dec. 3-7, 2012.

[11] Eduardo Cuervoy, Aruna Balasubramanianz, Dae-ki Cho,Alec Wolmanx, Stefan Saroiux, Ranveer Chandrax, Paramvir Bahlx. MAUI: Making Smartphones Last Longer with Code Offload.MobiSys, June 15–18, 2010, San Francisco, California, USA 2010.

[12] Cong Shi, Mostafa H. Ammar, Ellen W. Zegura, and Mayur Naik. Computing in Cirrus Clouds: The Challenge of Intermittent Connectivity. Mobile Cloud Computing (MCC) - ACM SIGCOMM 2012.

[13] Chi-Jen Wu, Jan-Ming Ho and Ming-Syan Chen.Time-Critical Event Dissemination in Geographically Distributed Clouds. IEEE INFOCOM Workshop on Cloud Computing 2011.

[14] Tianyi Xing, Hongbin Liang,Dijiang Huang, Lin X. CaiGeographic-based Service Request Scheduling Model for Mobile Cloud Computing. IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications, 2012.

[15] Yunqi Ye, Nisha Jain, Longsheng Xia, Suhas Joshi, I-Ling Yen, Farokh Bastani. A Framework for QoS and Power Management in a Service Cloud Environment with Mobile Devices. 2010 Fifth IEEE International Symposium on Service Oriented System Engineering.

[16] Yunqi Ye, Liangliang Xiao, I-Ling Yen, Farokh Bastani. Leveraging Service Clouds for Power and QoS Management for Mobile Devices. IEEE 4th International Conference on Cloud Computing 2011.